Dynamic programming

การโปรแกรมเชิงพลวัต (Dynamic programming) เป็นเทคนิควิธีที่ใช้ในการแก้ปัญหาเพื่อหาคำตอบ โดยมีแนวคิดในการแก้ปัญหาด้วยการแบ่งปัญหาออกเป็นปัญหาย่อย จากนั้นจะนำเอาคำตอบที่ได้จากปัญหาย่อยเหล่านั้นมารวมกันเพื่อให้ได้คำตอบในการแก้ปัญหา ตัวอย่างโจทย์ที่เหมาะกับการแก้ปัญหาด้วยวิธี Dynamic programming เช่น การแลกเหรียญ การหาเส้นทางสั้นสุด การหาลำดับย่อยร่วมกันที่ยาวสุด การเลือกของที่ได้มูลค่ารวมสูงสุด การหาลำดับการคูณเมทริกซ์เพื่อให้สามารถหาผลคูณได้เร็วสุด เป็นต้น ซึ่งจะเห็นได้ว่าโจทย์ที่เหมาะกับการแก้ปัญหาด้วยวิธี Dynamic programming จะต้องมีสมบัติดังนี้

1. คำตอบที่ดีที่สุดของปัญหาเกิดจากการรวมกันของคำตอบจากปัญหาย่อยๆ
2. ปัญหาย่อยจะมีการซ้อนทับกันหรือมีการหาคำตอบของปัญหาย่อยซ้ำๆ กัน

การใช้เทคนิค Dynamic programming ในการแก้ปัญหาสามารถแบ่งได้เป็น 2 วิธีหลักๆ ในการจัดเก็บคำตอบที่ได้จากปัญหาย่อยเพื่อที่จะสามารถนำคำตอบของปัญหาย่อยกลับมาใช้ใหม่ได้ คือ

1. Tabulation: Bottom Up Dynamic Programming
2. Memoization: Top Down Dynamic Programming

ขั้นตอนในการแก้ปัญหาด้วยเทคนิค Dynamic programming แบ่งได้ดังนี้

- นิยามปัญหาย่อย Subproblem
- กำหนดวิธีการรวมคำตอบของปัญหาย่อยเพื่อให้ได้คำตอบของปัญหาใหญ่ (Recurrence)
- กำหนด Basecase และหาคำตอบของ Basecase

Fibonacci number

0 1 1 2 3 5 8 13 21 34 ...

นิยาม Fibonacci number ตัวที่ n แทนด้วย F_n ซึ่งจะหาได้ดังนี้

$F_0 = 0$

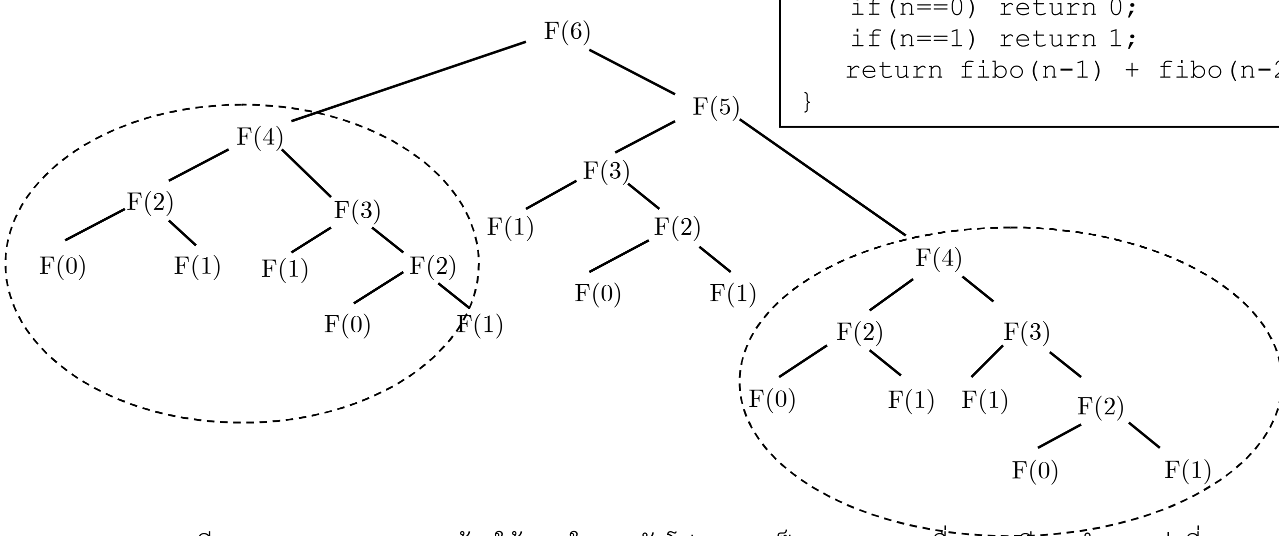
$F_1 = 1$

$F_n = F_{n-1} + F_{n-2}$

กรณีหา F_6 จะได้ว่ามีค่าเท่ากับ $F_5 + F_4$ โดยที่ $F_5 = F_4 + F_3$

เมื่อเขียนเป็นแผนผังการทำงานจะได้เป็นดังนี้

```
int fibo(int n) {
   if(n==0) return 0;
   if(n==1) return 1;
  return fibo(n-1) + fibo(n-2);
}
```



การเขียนแบบ Recursive จะต้องใช้เวลาในการรันโปรแกรมเป็นเวลานาน เนื่องจากมีการคำนวณค่าที่เคยคำนวณแล้วหลายครั้ง เช่น จากรูปการคำนวณ F_6 จะมีการคำนวณ F_2 ถึง 5 ครั้ง

การทำงาน

1. นิยาม **Subproblem**

   ให้ $C[i]$ แทน ตัวเลขลำดับที่ i → โจทย์ต้องการอะไร ?

2. หา **Recurrence** ของ **Subproblem**

   ให้ i แทน ลำดับที่ในชุดตัวเลข Fibonacci number

   ดังนั้น $C[i] = C[i-1] + C[i-2]$

3. กำหนด **Basecase** → ตัวสุดท้าย

   $C[0] = 0, \quad C[1] = 1$

Fibonacci number แบบ Bottom Up Dynamic Programming

```
// Tabulated version to find Fibonacci number.
int dp[MAXN];

// base case
dp[0] = 0;
dp[1] = 1;

for (int i = 2; i< =n; i++)
{
    dp[i] = dp[i-1] + dp[i-2];
}
```

0	1	2	3	4	5								
0	1	1	2	3	5	...							

Fibonacci number แบบ Top Down Dynamic Programming

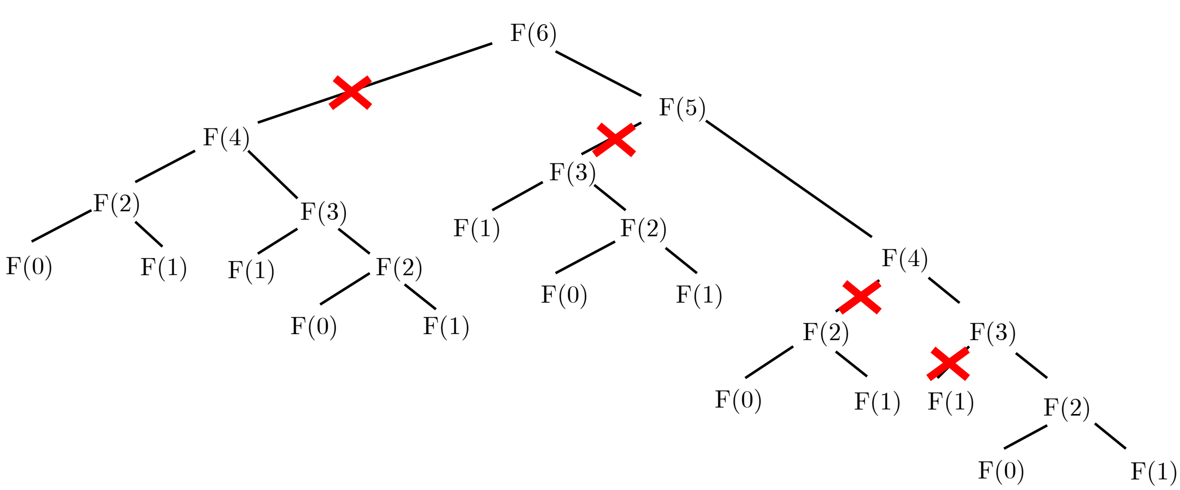
```
// Memoized version to find Fibonacci number.

// initialized to -1
int dp[MAXN]

// return Fibonacci(x)
int solve(int x)
{
    if (x==0)
        return 0;
    if (x==1)
        return 1;
    if (dp[x]!=-1)
        return dp[x];
    dp[x] = solve(x-1)+ solve(x-2);
    return (dp[x]);
}
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

ทำ Recursive ไม่ต่อ



Factorial number

นิยาม

$$n! = n\times(n-1)\times(n-2)\times\ldots\times 1$$

Recursive function

```
int factorial (int n) {
   if(n==1) return 1;
   return n * factorial(n-1);
}
```

การหา Factorial number ด้วย Dynamic programming

1. นิยาม Subproblem

   ให้ F[i] แทนค่า factorial ของ i

2. หา Recurrence ของ Subproblem

   F[i] = i × F[i-1]

3. กำหนด Basecase

int dp[MAX N]

*base case

F[1] = 1

```
for( int i = 2 ; i ≤ n ; i++ ){
      F[i] = F[i-1] + F[i-2]
}
```

Coin change-making problem

สมมติว่า มีเหรียญ 1, 4, 5, 10 บาท (d_1= 1, d_2= 4, d_3= 5, d_4= 10) ต้องการแลกเหรียญโดยจะต้องใช้จำนวนเหรียญให้น้อยที่สุด

- ต้องการแลกเงินจำนวน 7 บาท ใช้เหรียญอะไรบ้าง
- ต้องการแลกเงินจำนวน 8 บาท ใช้เหรียญอะไรบ้าง

การทำงาน

1. นิยาม **Subproblem**

   ให้ $C[p]$ แทนจำนวนเหรียญที่น้อยที่สุดที่ต้องใช้ในการแลกเงิน p บาท

2. หา **Recurrence** ของ **Subproblem**

   ให้ x แทน ค่าของเหรียญที่ถูกเลือกเพื่อให้ได้คำตอบที่ดีที่สุด

   ดังนั้น $C[p]=1+C[p-x]$

แต่อย่างไรก็ตามเราไม่รู้ค่า x และเพื่อให้ได้คำตอบ เราจะลองเปลี่ยนค่า x ไปเรื่อยๆ นั่นก็คือ $x=d_i$ โดยที่ d_i แทนมูลค่าของเหรียญที่ i และจะเลือก x ที่ทำให้มีจำนวนเหรียญรวมน้อยที่สุด

$$C[p]=\min_{i:d_i\le p}\{C[p-d_i]+1\}$$

3. **กำหนด Basecase**

   $C[0]=0$

แลกเหรียญ 8 บาท โดยมีเหรียญมูลค่า 1, 4, 5, 10 บาท เท่านั้น

จำนวนเงิน	ชนิดเหรียญ	จำนวนเหรียญที่ใช้
0	0	0
1	**กรณีที่ 1** มีเพียงเหรียญ 1 บาทจำนวน 1 เหรียญเท่านั้นจึงจะมีมูลค่าเท่ากับ 1 บาท $C[1]=1+C[1-1]=1+0=1$	1
2	**กรณีที่ 1** เลือก x เป็นเหรียญ 1 บาท จะได้ว่า 2-1 = 1 ซึ่งจากข้อมูลข้างบนจำนวนเงิน 1 บาท ใช้เหรียญน้อยที่สุด 1 เหรียญ ดังนั้นถ้า x ที่เลือกใช้เป็นเหรียญ 1 บาท จะต้องใช้เหรียญจำนวน 1+1 = 2 เหรียญ ในการแลกเงิน 2 บาท $C[2]=1+C[2-1]=1+1=2$	2
3	**กรณีที่ 1** เลือก x เป็นเหรียญ 1 บาท จะได้ว่า 3-1 = 2 ซึ่งจากข้อมูลข้างบนจำนวนเงิน 2 บาท ใช้เหรียญน้อยที่สุด 2 เหรียญ ดังนั้นถ้า x ที่เลือกใช้เป็นเหรียญ 1 บาท จะต้องใช้เหรียญจำนวน 1+2 = 3 เหรียญ ในการแลกเงิน 3 บาท $C[3]=1+C[3-1]=1+2=3$	3
4	**กรณีที่ 1** เลือก x เป็นเหรียญ 1 บาท จะได้ว่า 4-1 = 3 ซึ่งจากข้อมูลข้างบนจำนวนเงิน 3 บาท ใช้เหรียญน้อยที่สุด 3 เหรียญ ดังนั้นถ้า x ที่เลือกใช้เป็นเหรียญ 1 บาท จะต้องใช้เหรียญจำนวน 1+3 = 4 เหรียญ ในการแลกเงิน 3 บาท $C[4]=1+C[4-1]=1+3=4$ **กรณีที่ 2** เลือก x เป็นเหรียญ 4 บาท จะได้ว่า 4-4 = 0 ซึ่งจากข้อมูลข้างบนจำนวนเงิน 0 บาท ใช้เหรียญน้อยที่สุด 0 เหรียญ ดังนั้นถ้า x ที่เลือกใช้เป็นเหรียญ 4 บาท จะต้องใช้เหรียญจำนวน 1+0 = 1 เหรียญ ในการแลกเงิน 4 บาท $C[4]=1+C[4-4]=1+0=1$ *** จะเห็นได้ว่าจำนวนเหรียญที่น้อยที่สุดที่แลกเงิน 4 บาท จะต้องใช้กรณีที่ 2 คือ ใช้เหรียญ 4 บาทจำนวน 1 เหรียญ	1

5	**กรณีที่ 1** เลือก x เป็นเหรียญ 1 บาท จะได้ว่า 5-1 = 4 ซึ่งจากข้อมูลข้างบนจำนวนเงิน 4 บาท ใช้เหรียญน้อยที่สุด 1 เหรียญ ดังนั้นถ้า x ที่เลือกใช้เป็นเหรียญ 1 บาท จะต้องใช้เหรียญจำนวน 1+1 = 2 เหรียญ ในการแลกเงิน 5 บาท $C[5]=1+C[5-1]=1+1=2$ **กรณีที่ 2** เลือก x เป็นเหรียญ 4 บาท จะได้ว่า 5-4 = 1 ซึ่งจากข้อมูลข้างบนจำนวนเงิน 1 บาท ใช้เหรียญน้อยที่สุด 1 เหรียญ ดังนั้นถ้า x ที่เลือกใช้เป็นเหรียญ 4 บาท จะต้องใช้เหรียญจำนวน 1+1 = 2 เหรียญ ในการแลกเงิน 5 บาท $C[5]=1+C[5-4]=1+1=2$ **กรณีที่ 3** เลือก x เป็นเหรียญ 5 บาท จะได้ว่า 5-5 = 0 ซึ่งจากข้อมูลข้างบนจำนวนเงิน 0 บาท ใช้เหรียญน้อยที่สุด 0 เหรียญ ดังนั้นถ้า x ที่เลือกใช้เป็นเหรียญ 5 บาท จะต้องใช้เหรียญจำนวน 1+0 = 1 เหรียญ ในการแลกเงิน 5 บาท $C[5]=1+C[5-5]=1+0=1$ *** จะเห็นได้ว่าจำนวนเหรียญที่น้อยที่สุดที่แลกเงิน 5 บาท จะใช้กรณีที่ 3 คือ ใช้เหรียญ 5 บาทจำนวน 1 เหรียญ	1
6	**กรณีที่ 1** เลือก x เป็นเหรียญ 1 บาท จะได้ว่า 6-1 = 5 ซึ่งจากข้อมูลข้างบนจำนวนเงิน 5 บาท ใช้เหรียญน้อยที่สุด 1 เหรียญ ดังนั้นถ้า x ที่เลือกใช้เป็นเหรียญ 1 บาท จะต้องใช้เหรียญจำนวน 1+1 = 2 เหรียญ ในการแลกเงิน 6 บาท $C[6]=1+C[6-1]=1+1=2$ **กรณีที่ 2** เลือก x เป็นเหรียญ 4 บาท จะได้ว่า 6-4 = 2 ซึ่งจากข้อมูลข้างบนจำนวนเงิน 2 บาท ใช้เหรียญน้อยที่สุด 2 เหรียญ ดังนั้นถ้า x ที่เลือกใช้เป็นเหรียญ 4 บาท จะต้องใช้เหรียญจำนวน 1+2 = 3 เหรียญ ในการแลกเงิน 6 บาท $C[6]=1+C[6-4]=1+2=3$	2

	กรณีที่ 3 เลือก x เป็นเหรียญ 5 บาท จะได้ว่า 6-5 = 1 ซึ่งจากข้อมูลข้างบนจำนวนเงิน 1 บาท ใช้เหรียญน้อยที่สุด 1 เหรียญ ดังนั้นถ้า x ที่เลือกใช้เป็นเหรียญ 5 บาท จะต้องใช้เหรียญจำนวน 1+1 = 2 เหรียญ ในการแลกเงิน 2 บาท $C[6]=1+C[6-5]=1+1=2$ *** จะเห็นได้ว่าจำนวนเหรียญที่น้อยที่สุดที่แลกเงิน 6 บาท จะใช้กรณีที่ 1 หรือ 3 คือ ใช้เหรียญ 5 บาทจำนวน 1 เหรียญ เหรียญ 1 บาทจำนวน 1 เหรียญ	
7	**กรณีที่ 1** เลือก x เป็นเหรียญ 1 บาท จะได้ว่า 7-1 = 6 ซึ่งจากข้อมูลข้างบนจำนวนเงิน 6 บาท ใช้เหรียญน้อยที่สุด 2 เหรียญ ดังนั้นถ้า x ที่เลือกใช้เป็นเหรียญ 1 บาท จะต้องใช้เหรียญจำนวน 1+2 = 3 เหรียญ ในการแลกเงิน 7 บาท $C[7]=1+C[7-1]=1+2=3$ **กรณีที่ 2** เลือก x เป็นเหรียญ 4 บาท จะได้ว่า 7-4 = 3 ซึ่งจากข้อมูลข้างบนจำนวนเงิน 3 บาท ใช้เหรียญน้อยที่สุด 3 เหรียญ ดังนั้นถ้า x ที่เลือกใช้เป็นเหรียญ 4 บาท จะต้องใช้เหรียญจำนวน 1+3 = 4 เหรียญ ในการแลกเงิน 7 บาท $C[7]=1+C[7-4]=1+3=4$ **กรณีที่ 3** เลือก x เป็นเหรียญ 5 บาท จะได้ว่า 7-5 = 2 ซึ่งจากข้อมูลข้างบนจำนวนเงิน 2 บาท ใช้เหรียญน้อยที่สุด 2 เหรียญ ดังนั้นถ้า x ที่เลือกใช้เป็นเหรียญ 5 บาท จะต้องใช้เหรียญจำนวน 1+2 = 2 เหรียญ ในการแลกเงิน 7 บาท $C[7]=1+C[7-5]=1+2=3$ *** จะเห็นได้ว่าจำนวนเหรียญที่น้อยที่สุดที่แลกเงิน 7 บาท จะใช้กรณีที่ 1 หรือ 3 คือ ใช้เหรียญ 5 บาทจำนวน 1 เหรียญ เหรียญ 1 บาทจำนวน 2 เหรียญ	3

8	**กรณีที่ 1** เลือก x เป็นเหรียญ 1 บาท จะได้ว่า 8-1 = 7 ซึ่งจาก ข้อมูลข้างบนจำนวนเงิน 7 บาท ใช้เหรียญน้อยที่สุด 3 เหรียญ ดังนั้นถ้า x ที่เลือกใช้เป็นเหรียญ 1 บาท จะต้องใช้เหรียญจำนวน 1+3 = 4 เหรียญ ในการแลกเงิน 8 บาท $C[8]=1+C[8-1]=1+3=4$ **กรณีที่ 2** เลือก x เป็นเหรียญ 4 บาท จะได้ว่า 8-4 = 4 ซึ่งจาก ข้อมูลข้างบนจำนวนเงิน 4 บาท ใช้เหรียญน้อยที่สุด 1 เหรียญ ดังนั้นถ้า x ที่เลือกใช้เป็นเหรียญ 4 บาท จะต้องใช้เหรียญจำนวน 1+1 = 2 เหรียญ ในการแลกเงิน 8 บาท $C[8]=1+C[8-4]=1+1=2$ **กรณีที่ 3** เลือก x เป็นเหรียญ 5 บาท จะได้ว่า 8-5 = 3 ซึ่งจาก ข้อมูลข้างบนจำนวนเงิน 3 บาท ใช้เหรียญน้อยที่สุด 3 เหรียญ ดังนั้นถ้า x ที่เลือกใช้เป็นเหรียญ 1 บาท จะต้องใช้เหรียญจำนวน 1+3 = 4 เหรียญ ในการแลกเงิน 8 บาท $C[8]=1+C[8-5]=1+3=4$ *** จะเห็นได้ว่าจำนวนเหรียญที่น้อยที่สุดที่แลกเงิน 8 บาท จะใช้ กรณีที่ 2 คือ ใช้เหรียญ 4 บาทจำนวน 2 เหรียญ	2

คำถาม

ถ้ามีเหรียญในระบบเป็นเหรียญ 1 3 4 5 10

ต้องการแลกเงิน 7 บาท จะต้องใช้เหรียญน้อยที่สุดกี่เหรียญ และประกอบด้วยเหรียญอะไรบ้าง

Maximum subarray problem (Kadane's algorithm)

เป็นการหาผลรวมของลำดับย่อยของตัวเลขที่มีค่าผลรวมมากที่สุด โดยตัวเลขในลำดับย่อยจะต้องเป็นตัวเลขที่เรียงติดกันเท่านั้น

ชุดตัวเลข 4 -5 4 -3 4 4 -4 4 -5

ผลรวมของตัวเลขตั้งแต่ตัวที่ 1 ถึง 3 เท่ากับ 3 (4 + (-5) + 4)
ผลรวมของตัวเลขตั้งแต่ตัวที่ 5 ถึง 8 เท่ากับ 8 (4 + 4 + (-4) + 4)

4	**-5**	**4**	**-3**	**4**	**4**	**-4**	**4**	**-5**

1. นิยาม **Subproblem**

   ให้ $L(j)$ แทน ผลบวกที่มากที่สุดตั้งแต่ตำแหน่งก่อน j ถึงตำแหน่ง j

2. หา **Recurrence** ของ **Subproblem**

   ดังนั้น $$L(j) = \begin{cases} input(j) & ; input(j) > input(j) + \underbrace{L(j-1)}_{\text{ผลบวกก่อนหน้า}} \\ input(j) + L(j-1) & ; otherwise \end{cases}$$

   ผลบวกลำดับย่อยที่มากที่สุด คือ $\max(L(j))$

3. **กำหนด Basecase**

   $$L(0) = 0$$

จากข้อมูลต่อไปนี้

4	**-5**	**4**	**-3**	**4**	**4**	**-4**	**4**	**-5**

j	$input(j)$	$input(j)+L(j-1)$	$L(j)$	$\max(L(j))$
0	-	-	0 Base case	-
1	4	4	4	4
2	-5	-5 + 4 = -1	-1	max(4, -1) 4
3	4	3	4	4
4	-3	1	1	4
5	4	5	5	5
6	4	9	9	9
7	-4	5	5	9
8	4	9	9	9
9	-5	4	4	9

ตอบ 9

คำถาม

จากชุดตัวเลขเหล่านี้ จงหาผลรวมของลำดับย่อยของตัวเลขที่มีค่าผลรวมมากที่สุดว่ามีค่าเท่ากับเท่าใด

4	**-2**	**4**	**-3**	**2**	**4**	**-10**	**4**	**-5**

Longest increasing subsequence problem (LIS)

เป็นการหาจำนวนสมาชิกในลำดับย่อยที่มีจำนวนสมาชิกมากที่สุด โดยที่ตัวเลขในลำดับย่อยจะมีการเรียงลำดับจากน้อยไปมาก นอกจากนี้ตัวเลขในลำดับย่อยจะต้องเป็นตัวเลขเรียงลำดับตามโจทย์โดยไม่มีการสลับลำดับกันแต่ไม่จำเป็นที่จะต้องอยู่ติดกัน

ชุดตัวเลข 5 2 8 6 3 6 9 7

5	2	8	6	3	6	9	7

1. **นิยาม Subproblem**

   ให้ $L(j)$ แทน จำนวนสมาชิกของตัวเลขก่อนที่จะถูกโยงไปยังสมาชิกตัวที่ j

2. **หา Recurrence ของ Subproblem ของตัวเลขในลำดับย่อย (ที่เรียงจากน้อยไปมาก)**

   ให้ i แทน เลขตำแหน่งของตัวเลขที่จะถูกโยงไปยังตัวเลขในตำแหน่ง j
   ดังนั้น

   $$L(j) = 1 + \max\{L(i) : input(i) < input(j)\ \&\&\ i < j\}$$

แต่อย่างไรก็ตามเราไม่รู้ค่า i และเพื่อให้ได้คำตอบ เราจะลองเปลี่ยนค่า i ไปเรื่อยๆ และจะเลือกที่ทำให้มีจำนวนสมาชิกมากที่สุด $\max(L(j))$

3. **กำหนด Basecase**

   $$L(0) = 0$$

จากข้อมูลต่อไปนี้

5 2 8 6 3 6 9 7

j	$input(j)$	$L(i):\ i < j\ and\ input(i) < input(j)$	$L(j) = 1 + \max\{L(i)\}$	$\max(L(j))$
1	5	-	1 (first item)	1
2	2	-	1 (first item)	1
3	8	i=1 ; L(1) = 1 i=2 ; L(2) = 1	2	2
4	6	i=1 ; L(1) = 1 i=2 ; L(2) = 1	2	2
5	3	i=2 ; L(2) = 1	2	2
6	6	i=1 ; L(1) = 1 i=2 ; L(2) = 1 i=5 ; L(5) = 2	3	3
7	9	i=1 ; L(1) = 1 i=2 ; L(2) = 1 i=3 ; L(3) = 2 i=4 ; L(4) = 2 i=5 ; L(5) = 2 i=6 ; L(6) = 3	4	4
8	7	i=1 ; L(1) = 1 i=2 ; L(2) = 1 i=4 ; L(4) = 2 i=5 ; L(5) = 2 i=6 ; L(6) = 3	4	4

ตอบ 4

คำถาม

จากชุดตัวเลขเหล่านี้ จงหาจำนวนสมาชิกในลำดับย่อยที่มีจำนวนสมาชิกมากที่สุด

4 2 4 7 2 8 10 5 9

Find minimum jumps required to reach the destination

เป็นวิธีการหาจำนวนครั้งที่น้อยที่สุดของการกระโดดไปยังปลายทาง โดยที่ตัวเลขในลำดับอาจจะเป็นเลขจำนวนเต็มบวกหรือศูนย์ ซึ่งหมายถึงระยะทางหรือจำนวนช่องที่ไกลที่สุดที่สามารถกระโดดได้ (ไม่มีการกระโดดถอยหลัง) และเริ่มต้นให้อยู่ในช่องแรกเสมอ

ชุดตัวเลข 4 2 0 3 2 0 1 8

4	2	0	3	2	0	1	8

1. **นิยาม Subproblem**

   ให้ $L(j)$ แทน จำนวนครั้งที่ใช้ในการกระโดดมายังช่องที่ j

2. **หา Recurrence ของ Subproblem ของตัวเลขในลำดับย่อยที่เรียงจากน้อยไปมาก**

   ให้ j แทน ตำแหน่งก่อนกระโดดไปยังตัวเลขในตำแหน่ง $j+i$
   ดังนั้น

$$L(j+i)=\begin{cases} L(j)+1 & ;Jump\ to\ j+i\ \ (first\ \ time) \\ \min(L(j+i),L(j)+1) & ;otherwise \end{cases}$$

3. **กำหนด Basecase**

$$L(1)=0$$

จากข้อมูลต่อไปนี้

4 2 0 3 2 0 1 8

j	$input(j)$	i	update $L(j+i)$
1	4	i=1 กระโดดไปช่อง 2 i=2 กระโดดไปช่อง 3 i=3 กระโดดไปช่อง 4 i=4 กระโดดไปช่อง 5	L(1) = 0 L(2) = L(1)+1 = 1 (first time) L(3) = L(1)+1 = 1 (first time) L(4) = L(1)+1 = 1 (first time) L(5) = L(1)+1 = 1 (first time) L(6) = null L(7) = null L(8) = null
2	2	i=1 กระโดดไปช่อง 3 i=2 กระโดดไปช่อง 4	L(1) = 0 L(2) = 1 L(3) = min(L(3), L(2)+1) = min(1, 2) = 1 L(4) = min(L(4), L(2)+1) = min(1, 2) = 1 L(5) = 1 L(6) = null L(7) = null L(8) = null
3	0	i=0 ไม่มีกระโดด	L(1) = 0 L(2) = 1 L(3) = 1 L(4) = 1 L(5) = 1 L(6) = null L(7) = null L(8) = null
4	3	i=1 กระโดดไปช่อง 5 i=2 กระโดดไปช่อง 6 i=3 กระโดดไปช่อง 7	L(1) = 0 L(2) = 1 L(3) = 1

			L(4) = 1 L(5) = min(L(5), L(4)+1) = min(1, 2) = 1 L(6) = L(4)+1 = 2 (first time) L(7) = L(4)+1 = 2 (first time) L(8) = null
5	2	i=1 กระโดดไปช่อง 6 i=2 กระโดดไปช่อง 7	L(1) = 0 L(2) = 1 L(3) = 1 L(4) = 1 L(5) = 1 L(6) = min(L(6), L(5)+1) = min(2, 1+1) = 2 L(7) = min(L(7), L(5)+ 1) = min(2, 2) = 2 L(8)=null
6	0	i=0 ไม่มีกระโดด	L(1) = 0 L(2) = 1 L(3) = 1 L(4) = 1 L(5) = 1 L(6) = 2 L(7) = 2 L(8)=null
7	1	i=1 กระโดดไปช่อง 8	L(1) = 0 L(2) = 1 L(3) = 1 L(4) = 1 L(5) = 1 L(6) = 2 L(7) = 2 L(8)= L(7)+1 = 3 (first time)
8	8	-	L(1) = 0

			L(2) = 1 L(3) = 1 L(4) = 1 L(5) = 1 L(6) = 2 L(7) = 2 L(8)= 3

ตอบ 3

คำถาม

จงหาจำนวนครั้งที่น้อยที่สุดของการกระโดดไปยังปลายทางจากข้อมูลชุดตัวเลขที่อาจจะเป็นเลขจำนวนเต็มบวกหรือศูนย์ ซึ่งตัวเลขเหล่านี้หมายถึงระยะทางหรือจำนวนช่องที่ไกลที่สุดที่สามารถกระโดดได้ (ไม่มีการกระโดดถอยหลัง) และเริ่มต้นอยู่ในช่องแรกเสมอ

3	2	4	3	2	4	1	4	5

Longest common subsequence problem (LCS)

เป็นวิธีการหาความยาวที่มากที่สุดของข้อความย่อยที่ปรากฏอยู่ในทั้งข้อความที่ 1 และข้อความที่ 2 เช่น

ข้อความที่ 1 (v) SNOWY ความยาว (n) 5 ตัวอักษร

ข้อความที่ 2 (w) SUNNY ความยาว (m) 5 ตัวอักษร

1. สร้างอาเรย์ 2 มิติ

(n+1) * (m+1)

2. แถว 1 กับ col 1 = 0

		S	N	O	W	Y
	0	0	0	0	0	0
S	0	1	1	1	1	1
U	0	1	1	1	1	1
N	0	1	2	2	2	2
N	0	1	2	2	2	2
Y	0	1	2	2	2	3

ค่าแนวทแยง เท่ากัน

$$E_{i,j} = \begin{cases} E_{i-1,j-1} + 1 & , if\ \ v_i = w_j \\ \max(E_{i-1,j}, E_{i,j-1}) & , otherwise \end{cases}$$

ซ้าย บน

ข้อความที่ 1 (v) POLYNOMIAL ความยาว (n) 10 ตัวอักษร

ข้อความที่ 2 (w) EXPONENTIAL ความยาว (m) 11 ตัวอักษร

		P	O	L	Y	N	O	M	I	A	L
	0	0	0	0	0	0	0	0	0	0	0
E	0										
X	0										
P	0										
O	0										
N	0										
E	0										
N	0										
T	0										
I	0										
A	0										
L	0										

ข้อความย่อยที่ยาวที่สุดที่ปรากฏในทั้งสองข้อความ คือ PONIAL ความยาวเท่ากับ 6